To solve this problem, we need to figure out the optimal way to navigate from the center of a square grid to one of its corners, where the princess ('p') is located. The task is to print the shortest sequence of moves to reach her.

Problem Understanding:

- The grid is square, with a size N (odd), which means the center of the grid is at coordinates (N//2, N//2).
- The princess ('p') is located at one of the four corners of the grid, and we need to figure out the shortest path from the center to her.
- The valid moves are UP, DOWN, LEFT, and RIGHT.

Approach:

1. **Identify the center and corners**:
    - The center of the grid is always at position (N//2, N//2).
    - The four possible corners are:
        - Top-left: (0, 0)
        - Top-right: (0, N-1)
        - Bottom-left: (N-1, 0)
        - Bottom-right: (N-1, N-1)
2. **Find the position of the princess**:
    - We scan the grid to find the position of the princess ('p').
3. **Determine the path**:
    - We need to find the minimal number of steps from the center to the princess and generate a sequence of moves (UP, DOWN, LEFT, RIGHT) to reach her.
4. **Output the sequence**:
    - For each move, print the direction (e.g., "UP", "DOWN", etc.) on a new line.

Code Implementation:

```
def display_Path_to_Princess(N, grid):
    # Find the center position of the grid
    center = N // 2  # The center is at grid[N//2][N//2]

    # Locate the princess ('p') in the grid
```

```
princess_position = None
for i in range(N):
    for j in range(N):
        if grid[i][j] == 'p':
            princess_position = (i, j)
            break
    if princess_position:
        break

# Get the position of the princess
pr_x, pr_y = princess_position

# Current position of the player (at the center of the grid)
current_x, current_y = center, center

# To store the moves
moves = []

# Move vertically (UP or DOWN)
while current_x != pr_x:
    if current_x < pr_x:
        moves.append("DOWN")
        current_x += 1
    elif current_x > pr_x:
        moves.append("UP")
        current_x -= 1

# Move horizontally (LEFT or RIGHT)
while current_y != pr_y:
```

```
    if current_y < pr_y:
        moves.append("RIGHT")
        current_y += 1
    elif current_y > pr_y:
        moves.append("LEFT")
        current_y -= 1

# Output the sequence of moves
for move in moves:
    print(move)
```

Explanation of Code:

1. **Find Center**: We calculate the center of the grid as N // 2. This is where you start.
2. **Locate the Princess**: We loop through the grid and find the coordinates of the princess ('p').
3. **Generate Moves**: We move from the current position (center) to the princess' position:
    - If the center's x-coordinate is smaller than the princess' x-coordinate, we need to move DOWN. If it's larger, we move UP.
    - Similarly, we adjust the y-coordinate by moving LEFT or RIGHT depending on whether the current y-coordinate is smaller or larger than the princess' y-coordinate.
4. **Output the Moves**: We print each move on a new line.

Example Run:

Input:

5

--p--

Here, N = 5 and the princess is at position (2, 0), which is the bottom-left corner.

Output:

DOWN

DOWN

LEFT

LEFT

LEFT

How It Works:

- From the center (2, 2), we need to:
    - Move DOWN twice to reach row 4.
    - Move LEFT three times to reach column 0, where the princess is located.

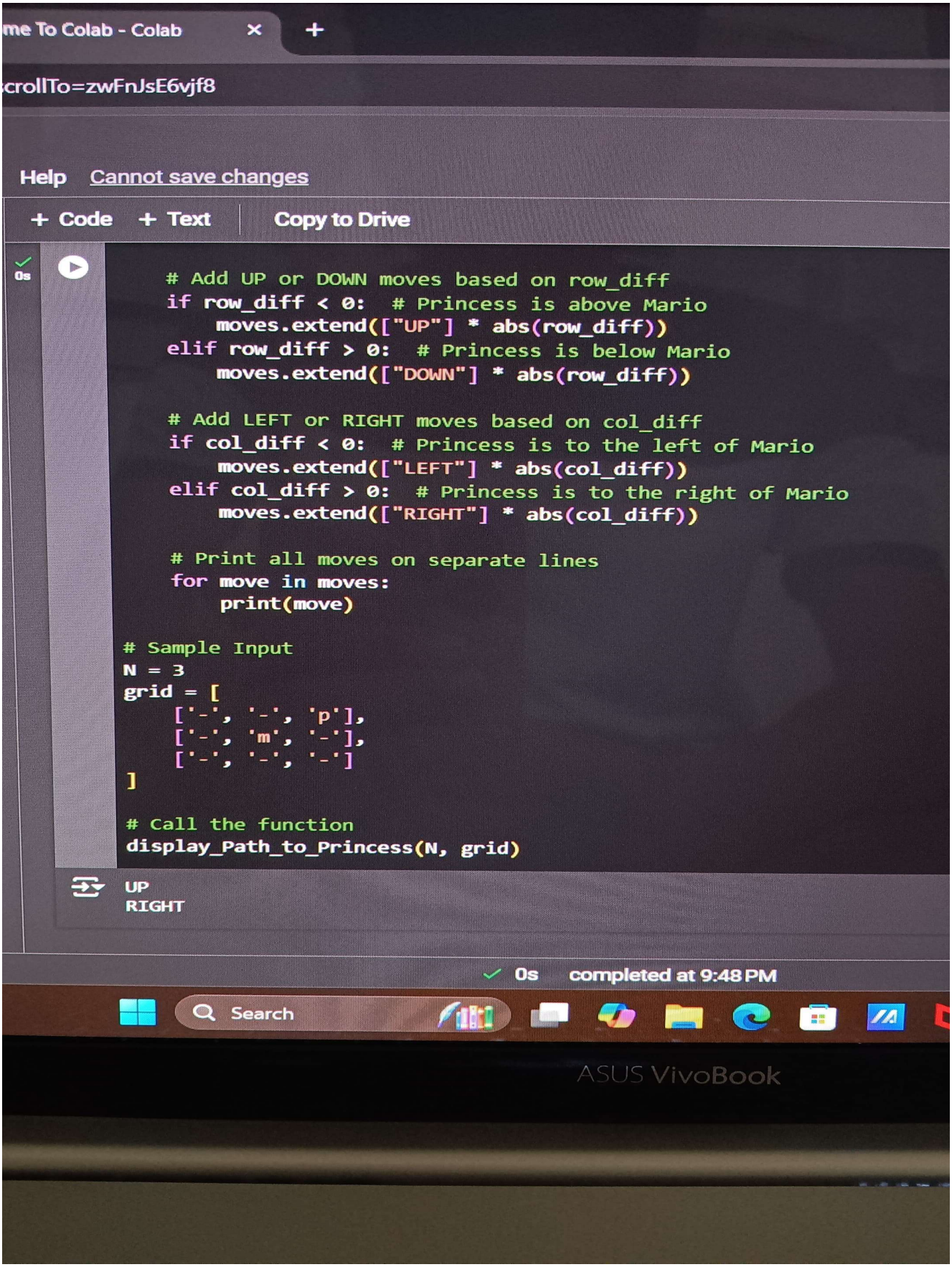
Time Complexity:

- We iterate through the grid once to find the princess, which takes O(N2)O(N^2).
- The move generation is O(N)O(N) in the worst case because we can move up to N−1N-1 steps either vertically or horizontally.
- Therefore, the overall time complexity is O(N2)O(N^2).

Edge Case Considerations:

- The princess is always at one of the corners, so no need to handle edge cases where she might not be found.
- The center position is always calculable since NN is always odd.

This solution ensures that the moves to rescue the princess are computed efficiently and correctly.


me To Colab - Colab
crollTo=zwFnJsE6vjf8
Help
Cannot save changes
+ Code
+ Text
Copy to Drive
0s
Add UP or DOWN moves based on row_diff
if row_diff < 0: # Princess is above Mario
moves.extend(["UP"] * abs(row_diff))
elif row_diff > 0: # Princess is below Mario
moves.extend(["DOWN"] * abs(row_diff))
# Add LEFT or RIGHT moves based on col_diff
if col_diff < 0: # Princess is to the left of Mario
moves.extend(["LEFT"] * abs(col_diff))
elif col_diff > 0: # Princess is to the right of Mario
moves.extend(["RIGHT"] * abs(col_diff))
# Print all moves on separate lines
for move in moves:
print(move)
# Sample Input
N = 3
grid = [
['-', '-', 'p'],
['-', 'm', '-'],
['-', '-', '-']
]
# Call the function
display_Path_to_Princess(N, grid)
UP
RIGHT
0s
completed at 9:48 PM
Search
ASUS VivoBook